

Gestion des exceptions et Threads

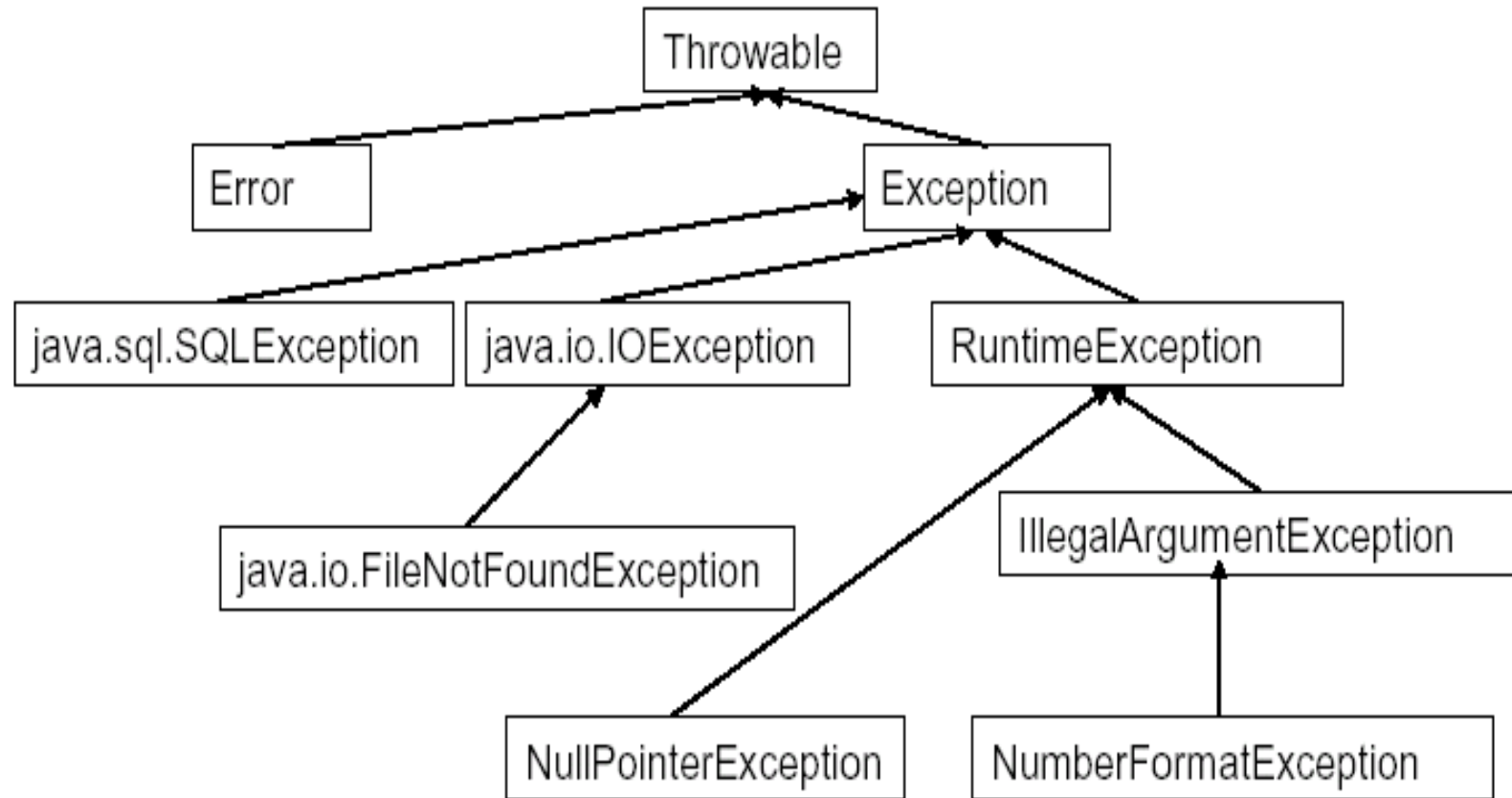
Prof Abdellah EZZATI

Gestion des exceptions

- Une exception est un signal indiquant que quelque chose d'exceptionnelle (comme une erreur, disque plein, division par zéro, ...) s'est produit.
- Elle interrompt le flot d'exécution normal du programme
- Plutôt que de compliquer le code du traitement normal, on traite les conditions anormales à part.
- Le traitement « normal » apparaît ainsi plus simple et plus lisible.
- Résoudre les problèmes d'exception soit :
 - ➔ En envoyant un message
 - ➔ Mettre fin au programme
 - ➔ Revenir en arrière
 - ➔ ...

Gestion des exceptions

- Toute exception en Java est un objet instancier d'une sous classe de la classe `Exception`
- Arbre des exceptions



Gestion des exceptions

→ Quelques exceptions prédéfinies :

→ Division par zéro : `ArithmeticException`

→ Référence nulle : `NullPointerException`

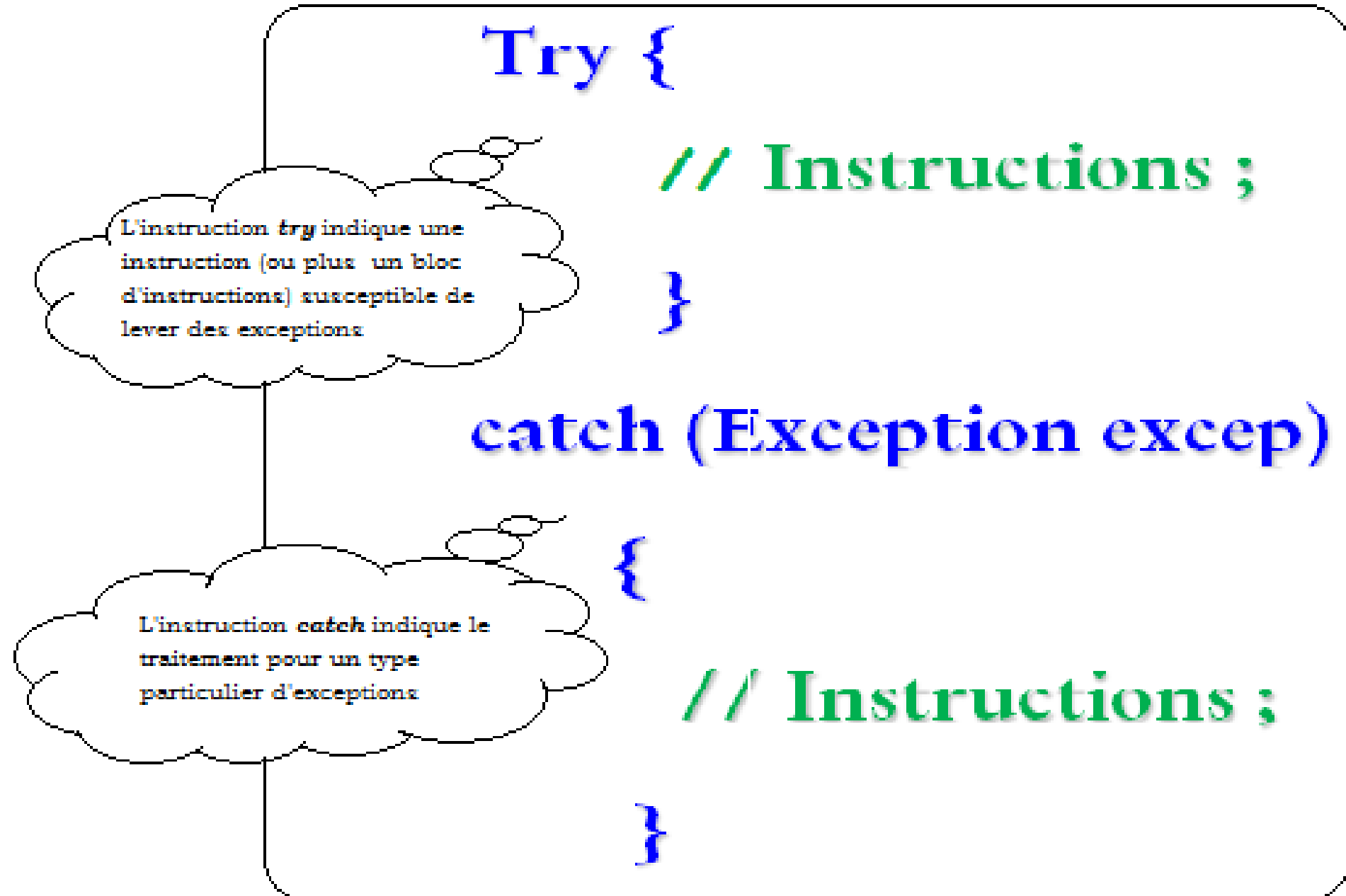
→ Tentative de forçage de type illégale : `ClassCastException`

→ Tableau de taille gative : `NegativeArraySizeException`

☐☐☐→Dépassement de dimension: `ArrayIndexOutOfBoundsException`

→ ...

Gestion des exceptions : `try` et `catch`



Gestion des exceptions

Exceptions de type RuntimeException

- Les exceptions de type RuntimeException correspondent à des erreurs qui peuvent survenir dans toutes les portions du codes:
 - ➔ ArithmeticException : division par zéro (entiers), etc
 - ➔ IndexOutOfBoundsException : dépassement d'indice tableau.
 - ➔ NullPointerException : référence null alors qu'on attendait une référence vers une instance.
 - ➔ ...

Exemple :

```

class Inv{
    private int n;
    public Inv(int n){this.n=n;}
    public void inverse(){
        if(n!=0) System.out.println ("1/"+n+" = " +(1/(float)n)) ;
        else      System.out.println ("1/"+n+" = " +(1/n)) ; }
    }
public class TestExcept3
{ public static void main (String args[])
    { Inv invDEN= new Inv(0) ;
      Inv invDEN1= new Inv(3) ;
      try
      { invDEN1.inverse() ;
        .....
        invDEN.inverse() ;
      }
      catch (RuntimeException e)
      { System.out.println (" division par 0 ") ; }
    }
}

```

Gestion des exceptions : contrôlées

→ Pour définir une nouvelle exception, on crée une nouvelle classe sous la forme suivante :

```
class NomDeClassEception extends ExceptionDejaDefinie { }
```

→ Lorsque l'on veut lancer une exception, on utilise le mot clé **throw** suivi de l'exception à lancer : **throw new DivZero();**

→ La clause **try** s'applique à un bloc d'instructions correspondant au fonctionnement normal mais pouvant générer des erreurs.

→ La clause **catch** s'applique à un bloc d'instructions définissant le traitement d'un type d'erreur. Ce traitement sera lancé sur une instance de la classe d'exception passée en paramètre.

Gestion des exceptions : contrôlées

Le JDK définit de nombreuses exceptions :

- `IOException` : fin de fichier.
- `FileNotFoundException` : erreur dans l'ouverture d'un fichier.
- `ClassNotFoundException` : erreur dans le chargement d'une classe.
- ...

Toute exception contrôlée, du JDK, pouvant être émise dans une méthode doit être :

- soit levée dans cette méthode. Elle est alors lancée dans un bloc `try` auquel est associé un `catch` lui correspondant.
- soit être indiquées dans le prototype de la méthode à l'aide du mot clé `throws`.

Gestion des exceptions : Exemple 1

```
class Inverse
{ private int I ;
  public Inverse(int i) throws DivZero
  { if ( i==0) throw new DivZero(); I = i ;    }
  public void affiche()
  { System.out.println (" 1/" + I + " = " + (1/(float)I)) ; }
}
class DivZero extends Exception { }
public class TestExcept1
{ public static void main (String args[])
  { try
    { Inverse [] Tab;
      Tab= new Inverse[11] ;
      for(int i=-5;i<=11;i++)
      {Tab[i+5]=new Inverse(i) ;
        Tab[i+5].affiche() ;}
    }
    catch (DivZero e)
    { System.out.println (" Division par zero ") ;    }
  }
}
```

Gestion des exceptions : Exemple 2

```
class Fact{
    private int n;
    public Fact(int n)throws PasDef {
        if(n<0) throw new PasDef(); this.n=n;}
    public int factorial(int n)
    { if(n==0) return 1;
      else return (n*factorial(n-1)); }
    public void affiche(){
        System.out.println (n+"! = "+factorial(n)) ; }
}
class PasDef extends Exception { }
public class TestExcept2
{ public static void main (String args[])
  {
    try
    { Fact factDen1= new Fact(5); factDen1.affiche();
      ..... Fact factDen2= new Fact(-5); factDen2.affiche();
    }
    catch(PasDef e)
    { System.out.println (" pas definie pour les negatifs ") ; }
  }
}
```

Gestion des exceptions : personnalisées

→ Les exceptions personnalisées sont des sous-classe de la classe `Exception`.

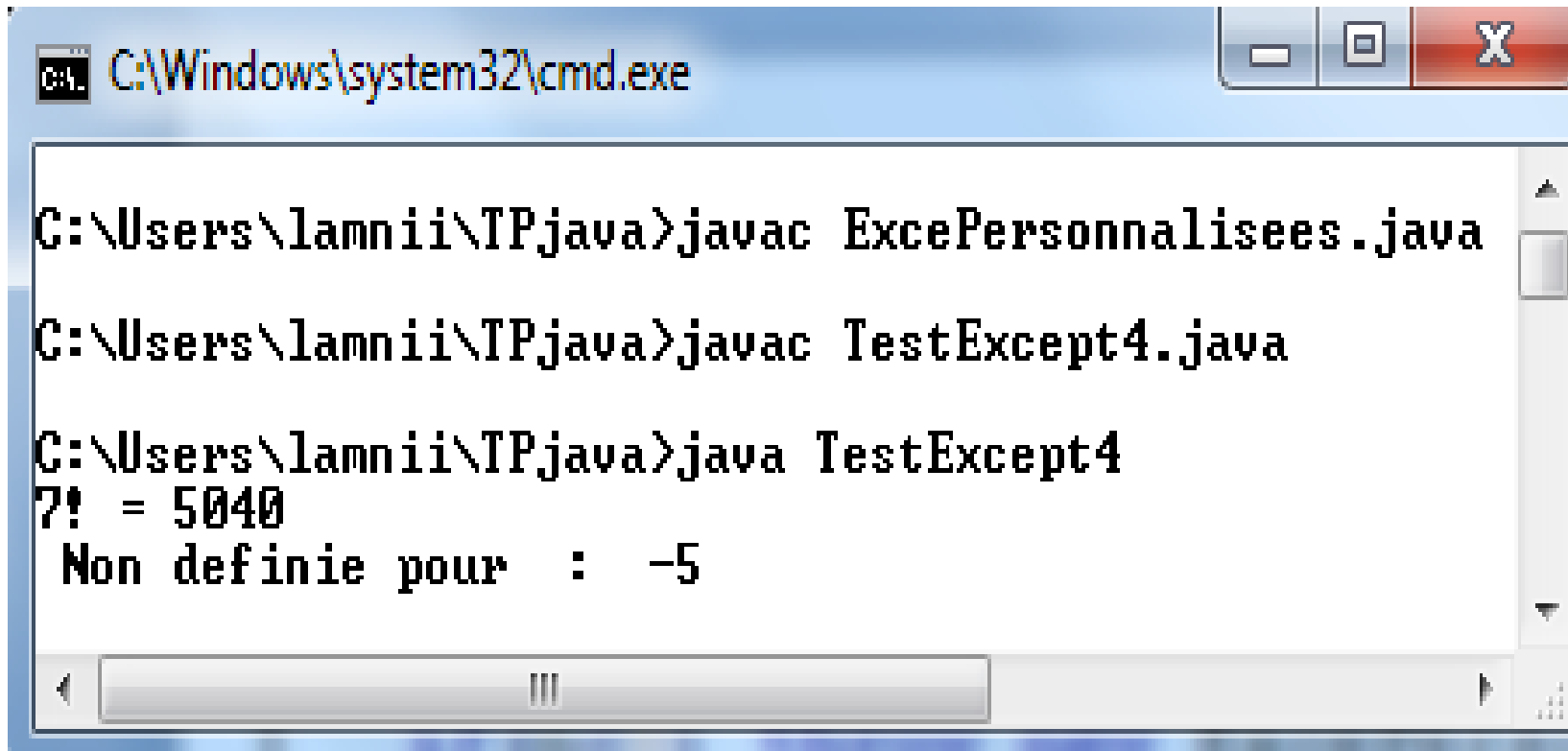
Exemple:

```
public class ExcePersonnalisees extends Exception {  
    private int n ;  
    public ExcePersonnalisees( int N ) {  
        n = N ; }  
    public String toString () {  
        return " Non definie pour : " + n ; }  
}
```

Gestion des exceptions : personnalisées

```
class Fact{
    private int n;
    public Fact(int n) throws ExcePersonnalisees {
        if(n<0) throw new ExcePersonnalisees(n); this.n=n;}
    public int factorial(int n)
    { if(n==0) return 1;
      else return (n*factorial(n-1)); }
    public void affiche(){
        System.out.println (n +"! = "+factorial(n)) ; }
}
public class TestExcept4
{ public static void main (String args[])
  {
    try
    { Fact factDen1= new Fact(7); factDen1.affiche();
      .....
      Fact factDen2= new Fact(-5); factDen2.affiche();
    }
    catch (ExcePersonnalisees e)
    { System.out.println (e) ; }
  }
}
```

Gestion des exceptions : personnalisées



```
C:\Windows\system32\cmd.exe

C:\Users\lamnii\TPjava>javac ExcePersonnalisees.java
C:\Users\lamnii\TPjava>javac TestExcept4.java
C:\Users\lamnii\TPjava>java TestExcept4
?! = 5040
Non definie pour : -5
```

Gestion des exceptions : finally

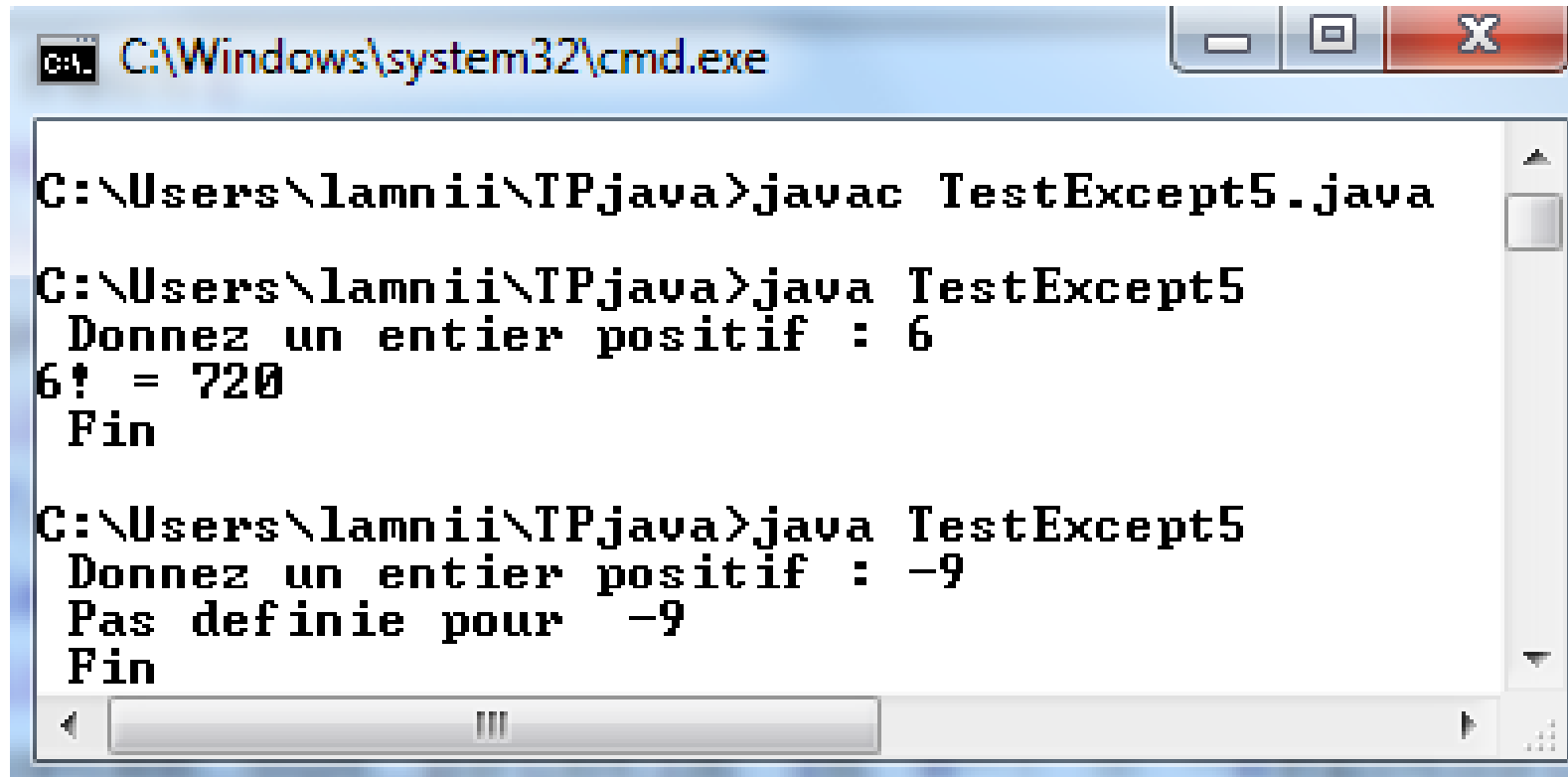
- La clause `finally` définit un bloc d'instruction qui sera exécuté même si une exception est lancée dans le bloc d'essai. Elle permet de forcer la bonne terminaison d'un traitement en présence d'erreur.
- Un blocs `finally` est exécutée quelle que soit le résultat du bloc `try`.
- `finally` pour ce qui s'exécute dans tous les cas

Exemple :

Gestion des exceptions : finally

```
class Fact{
    private int n;
    public Fact(int n)throws PasDef {
        if(n<0) throw new PasDef(); this.n=n;}
    public int factorial(int n)
    { if(n==0) return 1;
      else return (n*factorial(n-1)); }
    public void affiche(){
        System.out.println (n +"! = "+factorial(n)) ; }
}
class PasDef extends Exception { }
public class TestExcept5
{ public static void main (String args[])
  { int n; System.out.print(" Donnez un entier positif : ") ;
    n=Clavier.lireInt() ;
    try
    { Fact factDen1= new Fact(n); factDen1.affiche(); }
    catch(PasDef e)
    { System.out.println (" Pas definie pour " + n) ; }
    finally { System.out.println (" Fin ") ; }
  }
}
```


Gestion des exceptions : finally



```
C:\Windows\system32\cmd.exe

C:\Users\lannii\TPjava>javac TestExcept5.java

C:\Users\lannii\TPjava>java TestExcept5
  Donnez un entier positif : 6
6! = 720
  Fin

C:\Users\lannii\TPjava>java TestExcept5
  Donnez un entier positif : -9
Pas definie pour -9
  Fin
```

Exercice

- **Que fournit le programme suivant ?**

- `class Erreur extends Exception {}`
- `class A`
- `{ public A(int n) throws Erreur`
- `{ if (n==1) throw new Erreur() ; } }`

- `public class Chemin2`
- `{ public static void main (String args[])`
- `{ f(true) ; System.out.println ("apres f(true)");`
- `f(false) ; System.out.println ("apres f(false)"); }`
- `public static void f(boolean ret)`
- `{ try`
- `{ A a = new A(1) ; }`
- `catch (Erreur e)`
- `{ System.out.println ("** Dans f - exception Erreur ") ;`
- `if (ret) return ; }`
- `System.out.println ("suite f") ; }`
- `}`

Que fournit ce programme ?

```
class Erreur extends Exception {}
class Erreur1 extends Erreur {}
class Erreur2 extends Erreur {}
class A
{ public A(int n) throws Erreur
  { try
    { if (n==1) throw new Erreur1() ;
      if (n==2) throw new Erreur2() ;
      if (n==3) throw new Erreur() ;
    }
    catch (Erreur1 e)
    { System.out.println ("** Exception Erreur1 dans constructeur A") ;
    }
    catch (Erreur e)
    { System.out.println ("** Exception Erreur dans constructeur A") ;
      throw (e) ;
    }
  }
}

public class Redec1
{ public static void main (String args[])
  { int n ;
    for (n=1 ; n<=3 ; n++)
    { try
      { A a = new A(n) ;
      }
      catch (Erreur1 e)
      { System.out.println ("*** Exception Erreur1 dans main") ;
      }
      catch (Erreur2 e)
      { System.out.println ("*** Exception Erreur2 dans main") ;
      }
      catch (Erreur e)
      { System.out.println ("*** Exception Erreur dans main") ;
      }
      System.out.println ("-----") ;
    }
    System.out.println ("fin main") ;
  }
}
```

Les threads

- Un **thread** (tache en français, appelée aussi processus léger ou activité) est un fil d'instructions à l'intérieur d'un processus.
- Un **thread** n'est pas un objet. C'est un sous-processus qui exécute une série d'instructions d'un programme donné.
- Les programmes qui utilisent plusieurs threads sont dits **multithreadés**. Exemple (IHM)
- **Processus** : est un programme qui s'exécute et qui possède son propre espace mémoire.
- **Thread** : Un processus qui fonctionne comme un S.E. en lançant des sous-tâches internes au processus (multi-programmation). Ces sous-tâches sont nommées "flux d'exécution" ou Threads
- **Multithreading** : Windows, Solaris, MacOS,... supportent l'utilisation d'application contenant des threads: **Multithreading**.

Les threads

- La communication entre threads est plus rapide que celle entre processus, parce que les threads partagent un même espace de mémoire (de travail) entre eux, alors que les processus ont chacun son espace mémoire personnel.
- Un processus peut lancer plusieurs threads qui se partagent le même espace mémoire et peuvent donc se partager des variables

Exemples d'utilisation:

- interface graphique (IHM)
- le serveur réseau
- produit matricielle

Les threads : Création

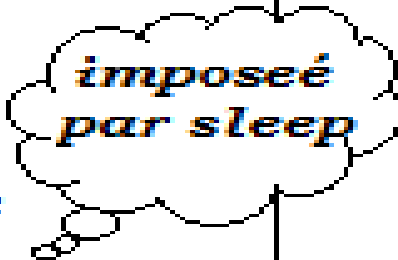
- Il existe deux moyens d'écrire des threads dans un programme. Les deux procédés passent par l'écriture d'une méthode `run()` décrivant les instructions que doit exécuter un thread.
- La méthode `run()` est soit :
 - déclarée dans une classe dérivée de la classe `Thread`
 - déclarée dans n'importe quelle classe qui doit alors implémenter l'interface.
- Une thread est lancée par appel d'une méthode `start()`

Les threads : Création

classe qui dérive de java.lang.thread	classe qui implante l'interface Runnable
<pre>class ThreadTest extends Thread { public void run() { // etc. } public ThreadTest(...) { // const. avec/sans args. } }</pre>	<pre>class ThreadTest implements Runnable { public void run() { // etc. } public ThreadTest(...) { // const. avec/sans args. } }</pre>
<pre>ThreadTest MonThread = new ThreadTest(...);</pre>	<pre>ThreadTest test = new ThreadTest(...); Thread t1 = new Thread(test);</pre>

Les threads : Exemple 1

```
public class DeuxThread {  
    public static void main(String args[ ]) {  
        .....  
        new Tache("Hola").start();  
        new Tache("Salut").start();  
    }  
}  
  
class Tache extends Thread {  
    public Tache(String str) {  
        .....  
        nom=str;  
    }  
  
    public void run() {  
        for (int i=0; i<13; i++) {  
            .....  
            try { System.out.println(nom);  
                .....  
                sleep((i*333));}  
            .....  
            catch (InterruptedException e) {}  
        }  
        System.out.println(nom + " est finie");  
    }  
    private String nom;  
}
```



*imposéé
par sleep*

Les threads : Exemple 1

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the following commands and output:

```
C:\Users\lamnii\TPjava>javac DeuxThread.java  
C:\Users\lamnii\TPjava>java DeuxThread  
Hola  
Hola  
Salut  
Salut  
Salut  
Hola  
Salut  
Hola  
Salut  
Hola  
Salut  
Hola  
Salut  
Hola  
Salut  
Hola  
Salut  
Hola  
Salut  
Hola  
Salut  
Hola  
Salut  
Hola  
Salut  
Hola  
Salut  
Hola  
Salut  
Hola est finie  
Salut est finie
```

The output consists of alternating "Hola" and "Salut" messages, followed by "Hola est finie" and "Salut est finie". The window includes standard Windows taskbar controls at the top right and a scrollbar on the right side.

Les threads : Exemple 2

```
public class TreadCalcul1
{ public static void main (String args[])
  { Somme s1=new Somme(5); Thread t1 = new Thread(s1); t1.start() ;
    Prod p1 =new Prod(5); Thread t2 = new Thread (p1); t2.start(); }
}
```

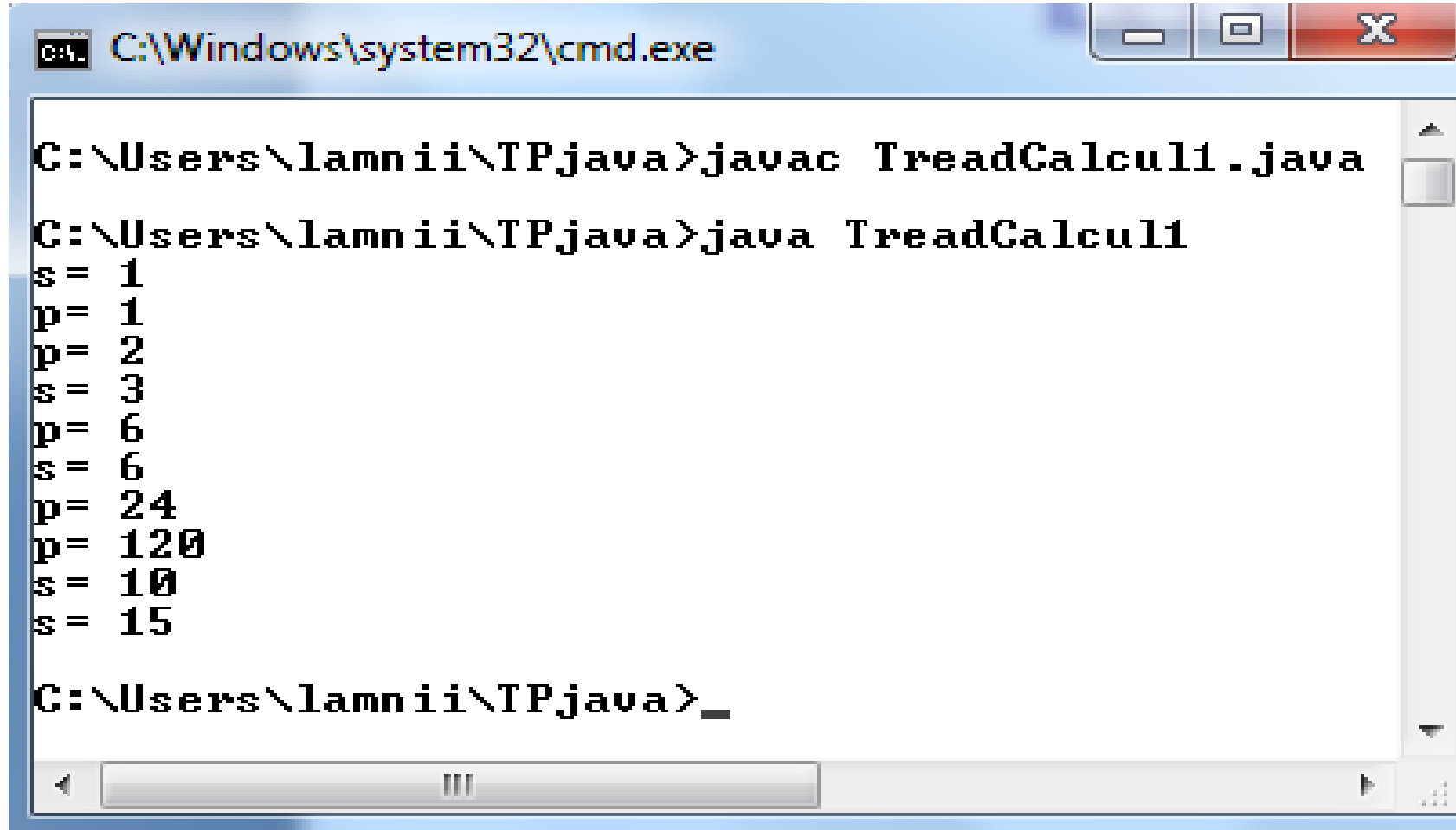
```
class Somme implements Runnable
{ private int nb ;
  public Somme (int nb)
  { this.nb = nb ; }
  public void run ()
  { try{ int i, s=0;
      for (i=1 ; i<= nb ; i++)
      { s=s+i; System.out.println ("s= " + s) ;
        Thread.sleep(3000); } }
    catch (InterruptedException e) {} }
}
```

classe "threadée"

```
class Prod implements Runnable
{ private int nb ;
  public Prod (int nb)
  { this.nb = nb ; }
  public void run ()
  { try{ int i, p=1;
      for (i=1 ; i<= nb ; i++)
      { p=p*i; System.out.println ("p= " + p) ;
        Thread.sleep(2000); } }
    catch (InterruptedException e) {} }
}
```

classe "threadée"

Les threads : Exemple 2



```
C:\Windows\system32\cmd.exe

C:\Users\lamnii\TPjava>javac TreadCalcul1.java
C:\Users\lamnii\TPjava>java TreadCalcul1
s = 1
p = 1
p = 2
s = 3
p = 6
s = 6
p = 24
p = 120
s = 10
s = 15

C:\Users\lamnii\TPjava>_
```

Les threads : Constructeurs

`public Thread();`

→ crée un nouveau Thread dont le nom est généré automatiquement (aléatoirement).

`public Thread(Runnable target);`

→ target est le nom de l'objet dont la méthode `run()` est utilisée pour lancer le Thread.

`public Thread(Runnable target, String name);`

→ on précise l'objet et le nom du Thread.

`public Thread(String name);` → on précise le nom du Thread.

Les threads : Méthodes

- ➔ `void run()`: La méthode contenant le code à exécuter par le Thread.
- ➔ `void start()`: démarrer un Thread. ➔ `void destroy()`: détruit le Thread courant
- ➔ `String getName()`: retourne le nom du Thread.
- ➔ `int getPriority()`: retourne la priorité du Thread.
- ➔ `void interrupt()`: interrompt le Thread. ➔ `void resume()`: redémarrer le Thread.
- ➔ `static boolean interrupted()`: teste si le Thread courant a été interrompu.
- ➔ `void join()`; ou `void join(long millis)`; ou `void join(long millis, int nanos)`: attendre la mort du Thread, ou après un `millis` de millisecondes, ou millisecondes plus nanosecondes.
- ➔ `void setPriority(int newPriority)`: changer la priorité du Thread.
- ➔ `static void sleep(long millis)`; ou `static void sleep(long millis, int nanos)`: mettre en veille le Thread pendant `millis` millisecondes ou millisecondes plus nanosecondes.
- ➔ `String toString()`: nom du thread, priorité, et le groupe à qui il appartient.

Les threads : Exemple 3

```
public class Somme implements Runnable
{ private int nb ;
  public Somme (int nb)
  { this.nb = nb ; }
  public void run ()
  { try{ int i, s=0;
      for (i=1 ; i<= nb ; i++)
      { s=s+i; System.out.println ("s= " + s) ;
        Thread.sleep(300); } }
    catch (InterruptedException e) {} }
}
```

```
public class Prod implements Runnable
{ private int nb ;
  public Prod (int nb)
  { this.nb = nb ; }
  public void run ()
  { try{ int i, p=1;
      for (i=1 ; i<= nb ; i++)
      { p=p*i; System.out.println ("p= " + p) ;
        Thread.sleep(150); } }
    catch (InterruptedException e) {} }
}
```

Les threads : Exemple 3

```
public class TreadCalcul2
{
    public static void main (String args[])
    {
        Somme s1=new Somme(5); Thread t1 = new Thread(s1); t1.start() ;
        Prod p1 =new Prod(5); Thread t2 = new Thread (p1); t2.start();
        try{t1.setPriority( 1 );
            Thread.currentThread().sleep( 200 );
            Thread.currentThread().sleep( 200 );
            System.out.println ("t1.getPriority()= " + t1.getPriority());
            System.out.println ("t1.getPriority()= " + t2.getPriority());
            System.out.println ("t1.getName()); = " + t1.getName());
            System.out.println ("t2.getName()); = " + t2.getName());
            t2.interrupt();
        }
        catch (InterruptedException e) {}
    }
}
```